一个有效的多边形裁剪算法*

刘勇奎[1+], 高 云[2], 黄有群[2]

[1](大连民族学院 计算机科学与工程系,辽宁 大连 116600)
[2](沈阳工业大学 信息科学与工程学院,辽宁 沈阳 110023)

An Efficient Algorithm for Polygon Clipping

LIU Yong-Kui[1+], GAO Yun[2], HUANG You-Qun[2]

[1](Department of Computer Science and Engineering, Dalian Nationalities University, Dalian 116600, China)
[2](College of Information Science and Engineering, Shenyang University of Technology, Shenyang 110023, China)
+ Corresponding author: Phn: 86-411-7612616 ext 4442, Fax: 86-411-7612616 ext 4442, E-mail: ykliu@dlnu.edu.cn
http://www.dlnu.edu.cn
Received 2001-07-31; Accepted 2001-10-17

Liu YK, Gao Y, Huang YQ. An efficient algorithm for polygon clipping. *Journal of Software*, 2003,14(4): 845~856.

Abstract: Polygon clipping is more often used than line clipping in practice, so it is the main subject in clipping research now. An efficient algorithm for polygon clipping which processes general polygons including concave polygons and polygons with holes inside is presented in this paper. This algorithm can be used to calculate not only intersection (clipping) but also set-theoretic differences and union of two polygons. It is based on some new techniques proposed in this paper. Firstly, singly linked lists are used as the data structure of this algorithm rather than doubly linked lists or trees as other algorithms use, so less memory space and running time are required. Secondly, the relationship between the entry and exit points on the two polygons is found, which, with the reasonable operations on the lists, reduces the times that the lists are traversed and allows the polygon to be input clockwise or counterclockwise. Lastly, finding and computing of intersection points is a main procedure. An efficient technique for finding and computing intersection points is presented, which makes the speed of the algorithm higher. At the end of this paper, the new algorithm is compared with the existing algorithms and the result shows that it uses less memory space and has higher speed than others.
Key words: computer graphics; concave polygons; polygon clipping; intersection calculation; singly linked list

摘 要: 多边形裁剪与线剪裁相比具有更广泛的实用意义,因此它是目前裁剪研究的主要课题.提出了一个多边形裁剪多边形的有效算法.其中的多边形都可以是一般多边形,既可以是凹多边形,也可以是有内孔的多边形.该算法不仅可以求多边形的“交”(多边形裁剪),而且可以求多边形的“并”和“差”.它是以所提出的一系列新方法和新技术为基础而形成的.首先,该算法使用单线性链表数据结构,与其他使用双链表或树结构的算法相比,具有

* Supported by the Science Foundation of National Nationalities Affair Committee of China (国家民委科技基金); the Science and Technology Foundation of Liaoning Province of China under Grant No.20022146 (辽宁省科技基金)

第一作者简介: 刘勇奎(1961－),男,辽宁沈阳人,博士,教授,主要研究领域为计算机图形学,图像处理.

占用空间少及处理速度快的特点;其次,找到了两个多边形之间进、出点之间的关系.再通过合理的数据结构处理,减少了算法对多边形链表的遍历次数,而且允许多边形既可以按顺时针方向也可以按逆时针方向输入.最后,判断和计算交点是裁剪算法的主要工作.提出了一个具有最少计算量的交点判断和计算方法,进一步加快了算法的运行速度.与其他同类算法进行了比较,结果表明,新算法具有最简单的结构和最快的执行速度.

关键词: 计算机图形学;凹多边形;多边形剪裁;交点计算;单链表结构

中图法分类号: TP391 **文献标识码**: A

在图形系统中,二维裁剪是最为基础、最为常用的操作之一.其典型的应用是在图形的消隐处理等各种三维图形的处理以及各种排料算法等求交操作之中.对裁剪算法的研究主要集中在裁剪直线和裁剪多边形两方面.在实用中,多边形裁剪与线剪裁相比具有更高的使用率,因此它是目前裁剪研究的主要课题.多边形裁剪用于裁剪掉被裁剪多边形(又称为实体多边形)位于窗口(又称为裁剪多边形)之外的部分.多边形愈复杂,其裁剪算法就愈难以实现.现有的解决方案或者局限于某一类多边形,或者结构复杂、时间消耗大.对于特殊情况已有几种有效的算法:Sutherland-Hodgeman[1]、梁-Barsky[2]、Foley[3]、Maillot[4]、Andereev[5]等算法要求裁剪多边形是矩形的.而在实际中,只有对于一般多边形的裁剪才有普遍意义,而且更实用.Montani 和 Re[6],Reppaport[7]以及 Sechrest 和 Greengberg[8]等人曾提出过对于一般多边形的裁剪算法,其中有些算法允许多边形具有孔洞.而普遍认为,在这类算法中只有 Weiler 算法[9]和近年出现的 Vatti 算法[10]及 Greiner-Hormann 算法[11]可以在合理的时间内处理一般的情况.其中,Weiler 算法使用的是树形数据结构,而 Vatti 算法和 Greiner-Hormann 算法使用的是双线性链表数据结构,所以后两者在复杂性及运行速度方面都优于前者.

本文提出了一个新的多边形裁剪多边形的有效算法.其中的裁剪多边形和被裁剪多边形都可以是一般多边形,既可以是凹多边形,也可以是有内孔的多边形.该算法只使用单线性链表数据结构,所以具有数据结构简单、占用空间少的特点,而且无须事先规定以什么方向输入图形的顶点.另外,该新算法使用了一个新的具有最少计算量的交点判断和计算方法,进一步加快了算法的运行速度.算法最终通过简单的遍历线性链表,可以得到每一个输出多边形.本文最后对新算法与目前被认为最有效的 Vatti 算法和 Greiner-Hormann 算法进行了各方面的比较.比较的结果表明,新算法优于上述两种算法.

本文第 1 节介绍有关多边形裁剪的一些基本概念及术语的定义.第 2 节给出了新算法的数据结构.第 3 节提出并描述了新算法.新算法所使用的交点判断和计算方法在第 4 节加以论述.第 5 节讨论了两多边形的边重合或两多边形在顶点处相交的特殊情况的处理.第 6 节对该新算法与 Vatti 算法和 Greiner-Hormann 算法在占用存储空间和执行速度方面进行了比较.第 7 节给出了新算法优于现有同类算法的结论.

1 基本概念与定义

为了便于下面对算法的讲解,本节首先介绍有关多边形裁剪的一些基本概念及术语.

(1) 多边形的边的方向与内外区域的关系.

如果多边形的边的方向是顺时针的(即多边形的顶点是以顺时针的顺序输入的),则在沿着多边形的边走时,右侧区域为多边形的内部;相反,如果多边形的边的方向是逆时针的,则在沿着多边形的边走时,左侧区域为多边形的内部.对于具有孔洞的多边形,只要把内孔边界和外边界以相反的方向表示,由上面的规则判断多边形的内部仍然适用.

(2) 进点和出点的定义.

设 I 是多边形 S 和 C 的一个交点,如果 S 沿着 S 的边界的方向在 I 点从 C 的外部进入 C 的内部,则称 I 为对于 C 的一个进点.反之,如果 S 在 I 点从 C 的内部出到 C 的外部,则称 I 为对于 C 的一个出点.

例如,对于如图 1 所示的多边形 C 和 S 及其交点 I,若 S 的方向为逆时针方向 $S_1\rightarrow S_2\rightarrow S_3\rightarrow S_4\rightarrow S_5$,则 $I_5I_1I_3$ 是对于 C 的进点,$I_4I_2I_6$ 是对于 C 的出点.如果 S 的方向为顺时针方向 $S_5\rightarrow S_4\rightarrow S_3\rightarrow S_2\rightarrow S_1$,则对于 C 来说,$I_2I_4I_6$ 是进点,$I_1I_5I_3$ 是出点.

(3) 进点和出点的判定.

假设多边形 S 的一条边 S_iS_{i+1} 与另一多边形 C 有交点.当点 S_i 是 C 的外点时,则沿着 S 的走向,边 S_iS_{i+1} 与 C 的第一个交点 I 必是 C 的进点;而当 S_i 是 C 的内点时,I 必是 C 的出点.由于沿着 S 的边界对于 C 的进点和出点是交替出现的(两多边形的边重合或者两多边形在顶点处相交的情况除外.这类特殊情况的处理将在第 5 节进行讨论),所以,只需判断第 1 个交点是进点还是出点,其他交点的进出性则可依次确定.

对于一个多边形裁剪另一个多边形的过程,就是求两个多边形的相交区域(我们称其为结果多边形或输出多边形).结果多边形是由实体多边形位于裁剪多边形内的边界和裁剪多边形位于实体多边形内的边界组成的.

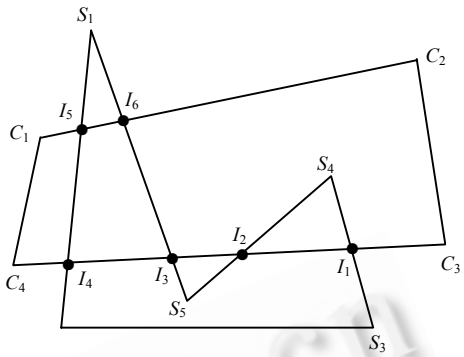


Fig.1 Definition of the entry and exit points

图 1 进点和出点的定义

2 新算法的数据结构

多边形裁剪算法需要一个适当的数据结构来存储多边形及交点,并能够在其上进行正确的操作.在 Weiler 的算法中,输入多边形组成一个树形结构.Greiner-Hormann 算法采用双向链表的结构,每个多边形由一个双向链表来表示.每找到一个交点,就将其分别插入到实体多边形和裁剪多边形的两个双向链表中.Greiner-Hormann 算法使用了线性链表,与 Weiler 算法的树形结构相比降低了数据结构的复杂性.本文的算法采用单链表来表示所有的多边形(输入和输出),与 Greiner-Hormann 算法的双向链表结构相比,不仅由于少用了一个指针域而节省了存储空间,而且还进一步降低了数据结构的复杂性.我们知道,在插入一个交点时,双向链表所需修改的指针数是单链表的 2 倍,因此,对单链表的操作不仅简单,而且也省时.

新算法的每个多边形由一个单链表来表示,单链表的每一个结点按序(多边形顶点输入的顺序)存储着多边形的一个顶点.最后一个结点的指针指向第 1 个结点(循环单链表).每个链表由一个头指针指向其第 1 个结点,实体多边形链表的第 1 个结点由头指针 HeadS 指示;裁剪多边形链表的第一个结点由头指针 HeadC 指示.结点的结构定义如下(其中 coordinates 表示坐标类型,用于存储顶点或交点的坐标值;pointer 表示指针类型):

```
Vertex={x, y: coordinates;
        inters, used: Boolean;
        next: pointer;
       }
```

交点的数据结构如下:

```
Intersection={x, y: coordinates;
              inters, used: Boolean;
              next1, next2: pointer;
              }
```

其中的指针域用于将交点分别插入到两个多边形的单链表中,第 1 指针域 next1 用于插入实体多边形链表;第 2 个指针域 next2 用于插入裁剪多边形链表.这样的数据结构定义使算法在求出一个交点时只需建立 1 个 Intersection(交点)类型的结点,并分别插入到两个多边形的单链表中,而不像 Greiner-Hormann 算法那样,要建立两个交点类型的结点,然后将每一个插入到一个多边形的链表中,而插入到两个链表中的这两个交点类型的结点之间也要用指针域 neighbor 彼此相连.

应该注意的是,实体多边形链表中的交点的进出性是对裁剪多边形而言的,而裁剪多边形链表中的交点的进出性则是对实体多边形而言的.

在这两个数据结构的定义中,布尔类型域 inters 用于区分该结点是否 Intersection(交点)类型的结点;used 域

用于有多个输出多边形时.所有交点的 used 域的初值都为 0,当一个交点被输出时,其 used 域被置为 1.

裁剪结果可能得到多个分立的输出多边形,因此需要设立一个指针链表 Out,其每个结点有一个指针域 polygon 指向一个输出多边形链表的第一个结点.该指针链表的结点结构如下:

```
Out={polygon: pointer;
     next: pointer;
    }
```

图 2 给出了对图 1 的多边形进行裁剪时,Greiner-Hormann 算法和新算法所使用的数据结构.可见,新算法的数据结构要比 Greiner-Hormann 算法简单得多.

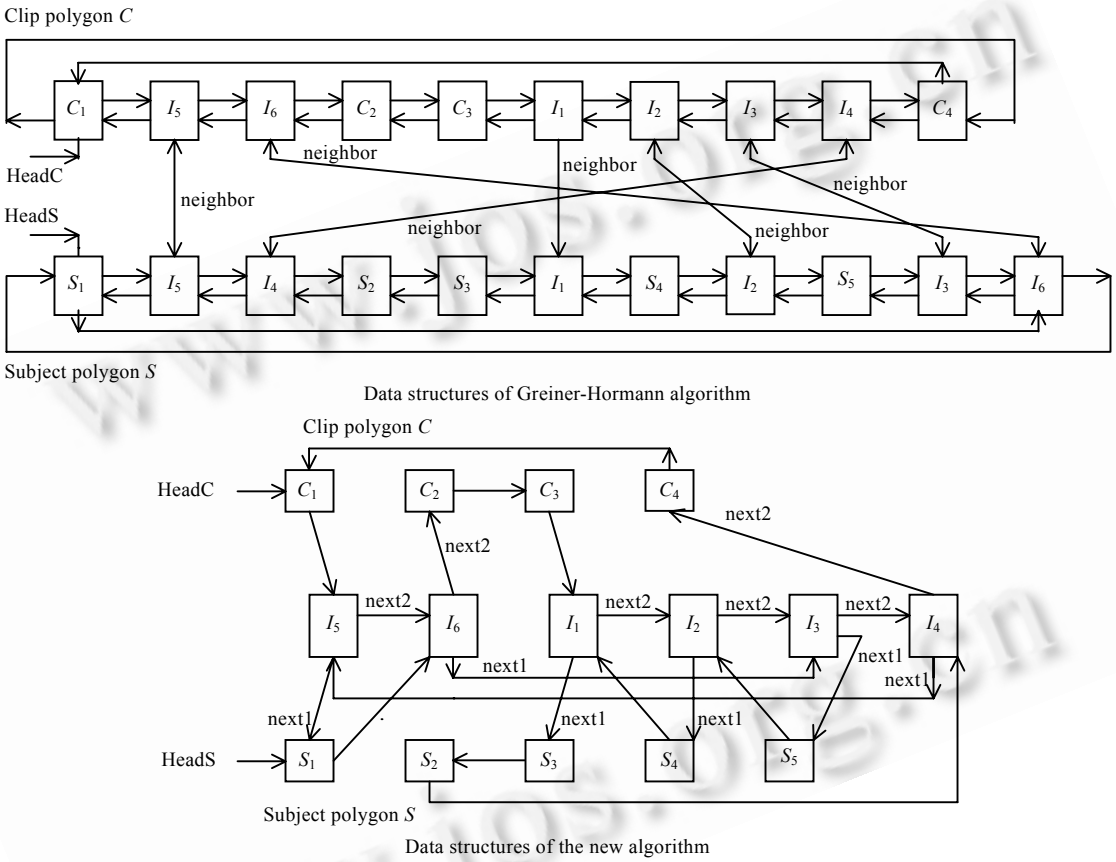


Fig.2 The data structures of Greiner-Hormann algorithm and the new algorithm for the polygons in Fig.1

图 2 对于图 1 的多边形进行裁剪时,Greiner-Hormann 算法和新算法所产生的数据结构

3 新算法

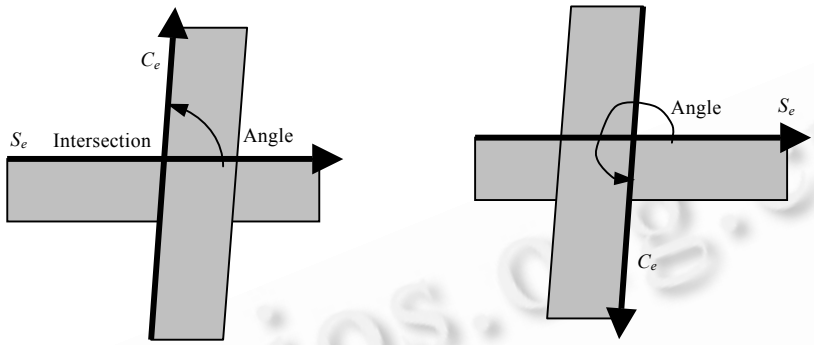
新算法分为 3 个阶段.第 1 个阶段,判断及计算第 1 个交点,并由其进出性判断两个多边形是否同向.如果不同向,则将裁剪多边形链表反向,然后将该交点插入到两个多边形的链表中.第 2 阶段,依次以实体多边形的每一个边对裁剪多边形进行直线裁剪操作,判断及计算其余交点,并以正确的顺序插入到两个多边形的链表中.第 3 阶段,遍历整个链表,输出最终结果.

我们以下面的引理开始算法第 1 阶段的描述.

引理 1. 如果两个相交多边形的边的取向相同(均为顺时针或逆时针方向),则对一个多边形是进点的交点对另一个多边形必是出点.

证明:设分别属于两个相交多边形 S 和 C 的两个相交边为 S_e 和 C_e,我们首先考虑两个相交多边形的边的取

向均为顺时针方向的情况.在这种情况下,多边形边的右侧为多边形内侧(在图 3 中由阴影表示).考虑两边之间的夹角,由于此夹角是由两边的相对位置决定的,所以我们可以将一个边的方向固定而讨论另一个边方向变化的各种情况.在图 3 中,设 S_e 边的方向是从左向右固定不变的,如果 C_e 的正向与 S_e 的正向的夹角在 0°~180° 之间(即 C_e 从下向上穿过 S_e(图 3a 的情况)),则由图 3 显然可见,C_e 经过交点从多边形 S 内走向 S 外,即该交点是对于多边形 S 的出点;而另一方面,边 S_e 则是经过该交点从多边形 C 外走向 C 内,即该交点是对于多边形 C 的进点.



(a) The angle of S_e with C_e is between 0° and 180° (b) The angle of S_e with C_e is between 180° and 360°
(a) S_e 和 C_e 的正向的夹角在 0° 和 180° 之间 (b) S_e 和 C_e 的正向的夹角在 180° 和 360° 之间

Fig.3 If the direction of the intersection edges (S_e and C_e) of the two polygons is clockwise, the intersection point is an entry point for one polygon and an exit point for the other polygon (the shadows represent the inside areas of the polygons)
图 3 当两个多边形的相交边 S_e 和 C_e 的取向均为顺时针方向时,对一个多边形是进点的交点对另一个多边形必是出点(阴影表示多边形内侧)

如果 C_e 的正向与 S_e 的正向的夹角在 180°~360° 之间(即 C_e 从上向下穿过 S_e(如图 3(b)所示),则由图显然可见,该交点对于多边形 S 是进点,而对于多边形 C 则是出点.这就是我们要证明的结论.

对于两个相交多边形的边的取向均为逆时针方向的情况,可用相同的方法证明该引理. □

根据该引理,对其中一个多边形求出一个进点或出点以后,在两个多边形方向相同的情况下,其对另一个多边形的进出性也就确定了.这样,如果两个多边形的方向相同,则在求出交点时只需判断和标记它对其中一个多边形是进点还是出点.它对另一个多边形的进出性则相反.而由第 1 节的讨论可知,由于沿着一个多边形的边界,在其上的进点和出点是交替出现的.所以只需标记第 1 个交点是进点还是出点,其他交点的进出性则可依次确定.最终我们得出一个结论:如果两个多边形的方向相同,则要标记所有交点对于两个多边形的进出性,只需标记任何一个多边形链表中的第 1 个交点的进出性即可(在后面的算法描述中,我们用变量 *Sin* 来标记实体多边形链表中的第 1 个交点对于裁剪多边形是否为进点).因此,新算法的第 1 步就要是判断两个多边形是否同向.如果不同向,则将裁剪多边形链表反向,使两个多边形的方向相同.

判断两个多边形是否同向,是通过判断一个交点(如第 1 个交点)对于两个多边形的进出性来完成的.如果该点对于实体多边形的进出性与对于裁剪多边形的进出性不同,则可知两个多边形取向相同;否则,两个多边形的取向相反.

新算法将交点的计算与进出性判断合成一步进行.当一个多边形的一个边对另一个多边形进行直线裁剪操作之后,如果有交点,即可根据交点在这个边上的排序的奇偶性来确定交点对另一个多边形的进出性.这样在计算交点的同时也确定了该交点的进出性.详细的描述见第 4 节.

下面是算法的第 1 部分的形式描述,其中指针变量 PS 和 PC 分别指向实体多边形链表和裁剪多边形链表中正在被处理的当前结点.另外,我们把由结点 PS↑和其下一个结点 PS↑.next↑定义的边简称为由 PS 指向的边.

```
PS=HeadS;
Repeat
    以 PS 指向的边与裁剪多边形进行直线裁剪操作(即求交点的操作,见第 4 节);
    if (上述直线裁剪操作有交点) then
```

{如图 2 所示,将每个交点(可能有多个)按其在该边上的顺序插入到实体多边形链表和裁剪多边形链表的对应相交边的两个结点之间;
由 *Sin* 标记插入到实体多边形链表中的第 1 个交点对于裁剪多边形的进出性,*Sin*=1 表示进;
令 PF 指向第 1 个交点结点,以备算法的第 3 阶段使用;
将 PC 指向该交点在裁剪多边形上的对应边;
以 PC 指向的边与实体多边形进行直线裁剪操作;
求出上述第 1 个交点对于实体多边形的进出性;
if 上述第 1 个交点对于实体多边形和裁剪多边形的进出性相同 then 逆转裁剪多边形的链表;
令 PS 指向实体多边形的下一个边;
转到算法的第 2 阶段;
}
令 PS 指向实体多边形的下一个边;
until PS=HeadS;
两个多边形无交点,算法结束;

在第 2 阶段,算法从第 1 阶段求出交点的那个实体多边形边的下一个边开始,用每一个实体多边形边与裁剪多边形求交点,并如图 2 所示,给每个交点建立一个包含该交点坐标的新的交点结点,然后将其插入到实体多边形链表和裁剪多边形链表的对应相交边的两个结点之间.例如,一个交点是由结点 PS↑和其下一个结点 PS↑.next↑所定义的实体多边形的边与由结点 PC↑和其下一个结点 PC↑.next↑所定义的裁剪多边形的边相交形成的,那么该交点结点就应该被插入到实体多边形链表的结点 PS↑和其下一个结点 PS↑.next↑之间,同时被插入到裁剪多边形链表的结点 PC↑和其下一个结点 PC↑.next↑之间.当一个边上有多个交点时,则以该边的方向为序将这些交点插入其中.例如,如果该边的方向是从左向右的斜线,则可按交点的 x 坐标的大小顺序插入这些交点.

在这个阶段,算法不需要标记交点的进出性,因为如前所述,算法只需在第 1 阶段用变量 *Sin* 来标记实体多边形链表中的第 1 个交点对于裁剪多边形的进出性,其余交点对于两个多边形的进出性便由如前所述的规律可知.下面是算法的第 2 部分的形式描述.

Repeat
以 PS 指向的边与裁剪多边形进行直线裁剪操作;
if (上述直线裁剪操作有交点) then 将每个交点按其在该边上的顺序插入到实体多边形链表和裁剪多边形链表的对应相交边的两个结点之间;
令 PS 指向实体多边形的下一个边;
until PS=HeadS;
转到算法的第 3 阶段;

在算法的第 3 阶段,通过遍历已插入交点结点的实体多边形和裁剪多边形链表来跟踪结果多边形的边界,最后产生输出多边形链表.

跟踪一个结果多边形的边界是以实体多边形链表中的一个进点(对于裁剪多边形)开始的.从该进点到实体多边形链表中的下一个交点(记为 N1)之间的实体多边形的边界全部是结果多边形的边界.N1 既是对于裁剪多边形的出点也是对于实体多边形的进点,因此从 N1 点开始到裁剪多边形链表中的下一个交点之间的裁剪多边形的边界全部是结果多边形的边界(如图 1 所示).输出这些边界.重复此过程,一直到回到实体多边形链表中的开始进点为止,便跟踪输出了一个结果多边形.在上述过程中,实体多边形链表中的进点(对于裁剪多边形)的 used 域被标记为 1,以表明从它开始的边界已经被输出过.“used 域”用于有多个结果多边形的情况.

算法第 3 阶段的遍历是以实体多边形链表的顺序进行的.从实体多边形链表中的第 1 个进点开始,如果该进点(当前进点)的 used 域为 0(表明它未被输出过),则将其置为 1,并执行上一段所述的跟踪过程输出一个结果多边形;如果该进点的 used 域为 1,则走到实体多边形链表的下一个进点,即该进点的下一个交点的下一个交点(如前所述,在多边形链表中交点的进出性是相隔分布的).以下一个进点为当前进点,重复此过程,一直到回到实

体多边形链表中的第 1 个进点为止.所有的进点都被访问过,所有的结果多边形也都被输出.至此,算法结束.下面是算法第 3 阶段的形式描述.

if *Sin*=0 then 令 PF 指向实体多边形链表中的下一个交点结点,以确保 PF 指向第 1 个进点;
PP=PF;
repeat
  if PP 所指的交点结点的 used 域为 0 then
    {PO=PP;
     建立一个新的输出多边形链表,并将指向该链表的头指针加入到指针链表 Out 的最后(在 polygon 域当中);
     repeat
       将 PO 所指的交点结点(一定是进点)的 used 域置为 1;
       将从 PO 所指的交点结点开始(用 next1 指针域)到下一个交点结点(记为 N1)之前的实体多边形链表中的结点加入到输出多边形链表的最后,并使 PO 指向 N1;
       将从 PO 所指的交点结点开始(用 next2 指针域)到下一个交点结点(记为 N2)之前的裁剪多边形链表中的结点加入到输出多边形链表的最后,并使 PO 指向 N2;
     until PO=PP
    }
  else 使 PP 指向实体多边形链表中的下一个进点结点(即下一个交点结点的下一个交点结点);
until PP=PF;
算法结束,输出多边形链表由指针链表 Out 的 polygon 域指出.

该算法不仅可以求多边形的“交”(多边形裁剪),而且稍加修改就可以求多边形的“并”和“差”.只要输出多边形从出点到入点的边(而不是上述的从入点到出点)即可得到多边形的“并”.要得到多边形的“差”也很简单,只需使两个多边形一个顺时针取向,另一个逆时针取向即可.

对于有内孔的多边形,只要把内孔边界和外边界以相反的方向表示,则第 1 节的多边形的边的方向与内外区域的关系仍然适用.此时,只判断实体多边形和裁剪多边形的外边界方向即可:若方向相同,则不必调整;若方向相反,则把裁剪多边形的链表反向.然后,分别求出实体多边形的内、外边界与裁剪多边形的内、外边界的交点,并把交点插入到相应的数据结构中.最后遍历所有交点求出输出结果多边形.

为了使算法能够裁剪有内孔的多边形,只需对上述算法进行少量的修改.用一个链表来表示有内孔的多边形时会多出一条边,如图 4 中的边 C_1C_9.为了避免出现这种情况,我们采用如下的方法:将多边形链表的第 1 个结点换成上述交点结点的结构,即具有两个指针域 next1 和 next2.其中 next1 用于指向多边形的外边界的第 2 个结点,而 next2 用于指向多边形的内边界的第 1 个结点.内边界的第 1 个结点同样具有两个指针域 next1 和 next2,如果有第 2 个内孔,则由 next2 指向;如果没有,其 next2 指针域为空.这样,对多边形链表(可能包括多个边界链表)遍历的结束条件就不是回到链表的第 1 个结点了,而是回到其 next2 指针域为空的第 1 个结点.在算法中,每个边界的第 1 个结点都设一个头指针 Head 指向,自然区别于其他结点.

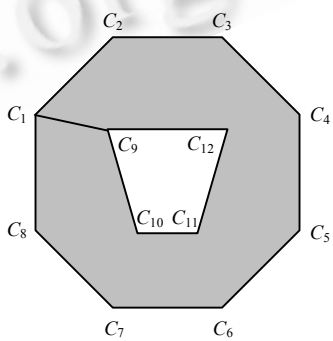


Fig.4 If a polygon with a hole is represented by a link, a superfluous edge C_1C_9 is added

图 4 用一个链表来表示有内孔的多边形时会多出一条边 C_1C_9

4 交点的判断与计算

由上述分析可知,多边形裁剪的交点判断和计算是以多边形窗口的线裁剪为基础的,因为在多边形裁剪中求交点时是用实体多边形的每一条边与裁剪多边形求交点,即为线裁剪.关于多边形窗口的线裁剪算法,目前已有几种有效的算法.

对于凸多边形窗口的线裁剪,一个著名的算法是由 Cyrus 和 Beck 提出的[12].它通过判断直线段的方向矢量与窗口边法矢量的点积是否大于 0 而将所有交点分为上、下两组.然后,分别取上组中的最小交点和下组中的最大交点,即为线段可见部分的端点.但该算法对于凹多边形窗口这样的分组是没有意义的.所以,Cyrus-Beck 算法只适用于凸多边形窗口的线裁剪.

对于一般多边形窗口的线裁剪,我们最近发表的文献[13]提出了一个有效的算法(暂称为斜率法).该算法首先要在被裁剪的直线或其延长线上选择一个固定点(X_f,Y_f).这个固定点应在多边形窗口的最左顶点之左(或最下顶点之下).然后,计算从该固定点到多边形各顶点连线的斜率(以下称其为顶点的斜率),公式如下:

$$vs_i=\frac{y_i-Y_f}{x_i-X_f},$$

其中 $v_i(x_i,y_i)$为第 i 个顶点.

这样就可以依次判断被裁剪直线的斜率是否在多边形的两个相邻顶点的斜率之间.若是,则被裁剪直线或其延长线与这两个相邻顶点所定义的边相交;否则不相交,如图 5 所示.如果相交,则计算交点.

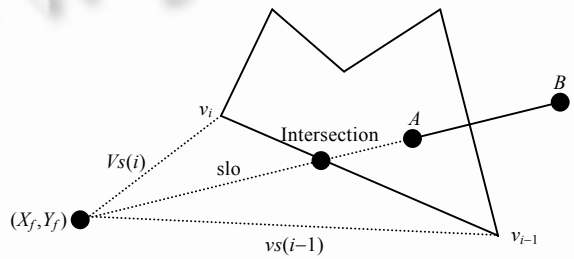


Fig.5 Finding intersection by the Slope Method (slo represents the slope of the line to be clipped)

图 5 斜率法的交点判断(slo 为被裁剪直线的斜率)

本文采用一个更有效的、适用于一般多边形的判断和计算交点的方法,我们称其为错切变换法.下面加以描述.

设多边形窗口 C 有 n 条边,C 的顶点 v_i 的坐标为(x_i,y_i),i=1,2,…,n;被裁剪线段为 L,其端点 A 和 B 的坐标分别为(x_a,y_a)和(x_b,y_b),如图 6 所示.

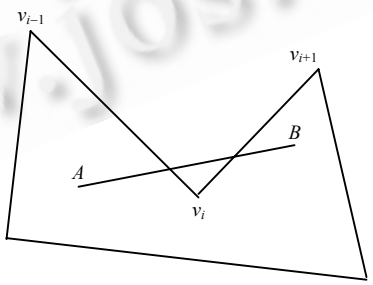


Fig.6 The polygon window and the line to be clipped

图 6 多边形窗口与被裁剪线段

下面以 $x_b\geq x_a$ 情况为例,记$\Delta x=x_b-x_a$,$\Delta y=y_b-y_a$,并假定$\Delta x\neq 0$,$\Delta y\neq 0$,否则 L 将与一条坐标轴平行,而这种情况无须错切变换,使执行过程更为简单.

在设计裁剪算法时,最主要的考虑是尽可能减少不必要的交点计算.本方法给出了很简单的判断条件,可以不计算落在窗口边的延长线上的非有效交点.对于落在被裁剪直线的延长线上的非有效交点,则只有当它被计

算出之后才能确定其是否有效(即在直线的两端点之间).

首先对 C 和 L 施加相同的错切变换,用沿 Y 轴方向的错切变换将 L 变换成与 X 轴平行(水平)的方向.沿 Y 轴方向的错切变换的矩阵是 $\begin{pmatrix}1 & d\\ 0 & 1\end{pmatrix}$,写成变换的分量形式是 $\begin{cases}x'=x\\ y'=x\cdot d+y\end{cases}$,其中 x'和 y'表示错切变换后的坐标值(下面均以这种形式表示变换后的点或坐标).显然,错切变换对这些点的 x 坐标没有影响,如图 7 和图 8 所示.设直线 L 与 Y 轴的交点为 $I(0,y_c)$,这里,$y_c=x_a\cdot d+y_a$.容易验证,经错切变换后,直线 AB 变为直线 $y=y_c$.

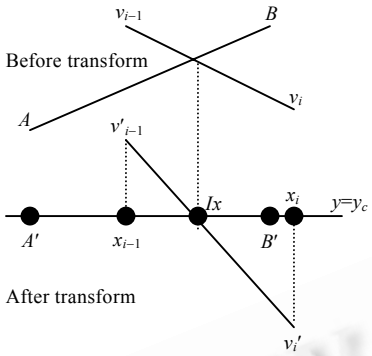


Fig.7 Alternating transform

图 7 错切变换

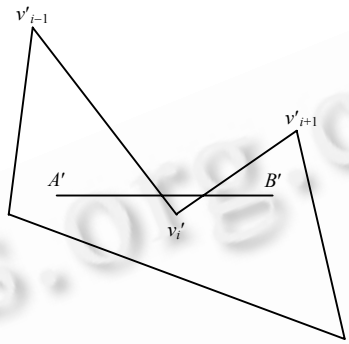


Fig.8 The clip polygon and the line after alternating transform from Fig.6

图 8 图 6 经错切变换后的裁剪多边形与被裁剪线段(变为水平线)

错切变换是一种仿射变换,它不能保留图形的度量性质,会引起图形形状的改变,但是直线之间的相对关系(相交关系)是不变的.由于对 C 和 L 进行错切变换后,L 变成了水平直线,所以很容易判断和计算交点.又由于变换前后的相交关系及其交点的 x 坐标都没有变化,因此在(错切变换后)求出交点以后只需对其 y 坐标进行反错切变换即可.从上面的错切变换公式可见,错切变换及其反变换的计算量是很小的.

经过错切变换,容易看出,C 的一条边与直线 AB 相交的条件是该边两顶点变换后的点 $v_{i-1}'(x_{i-1}',y_{i-1}')$和 $v_i'(x_i',y_i')$位于直线 $y=y_c$ 两侧,或至少一个变换后的顶点落在该直线上.我们先讨论前一种情况.这时,线段 $v_{i-1}'v_i'$ 与直线 $y=y_c$ 的交点的 x 坐标可计算如下:

$$Ix=x'_{i-1}+\frac{x'_i-x'_{i-1}}{y'_i-y'_{i-1}}(y_c-y'_{i-1})=x_{i-1}+\frac{x_i-x_{i-1}}{y'_i-y'_{i-1}}(y_c-y'_{i-1}). \tag{1}$$

上述过程可以形式化地描述如下:

(1) 计算$\Delta x,\Delta y,d,y_c,y_a'$和 y_b';

(2) 对每个 v_i,计算 $y_i'=x_i\cdot d+y_i$,(i=1,2,…,n);

(3) 令 $y_0'=y_n'$; j=1;

(4) 对每个 i($1\le i\le n$),依次作:

if ($y_{i-1}'>y_c$)AND ($y_i'<y_c$) OR ($y'_{i-1}<y_c$)AND ($y_i'>y_c$) then

{用式(1)计算交点的 x 坐标 Ix_j; $j=j+1$}

如上一节所述,新算法在第 1 次计算交点的同时也确定了该交点的进出性.具体的方法是,根据交点在被裁剪直线上的奇偶性来确定交点的进出性,奇序数交点是进点,偶序数交点是出点.由于被裁剪直线在错切变换后变成了水平直线,所以根据交点的 x 坐标的大小就可以决定交点在被裁剪直线上的顺序.如果求出一个交点的 x 坐标之后,经过判断发现该交点位于被裁剪直线的延长线上而不在直线的两端点 A'和 B'之间,则该交点不被插入到多边形链表中,但是去掉该交点可能对被裁剪直线上其他交点顺序的奇偶性产生影响.我们的办法是,如果被裁剪直线的方向是从A'到B',则对A'侧延长线上的交点不保留,但只累计个数,以保持有效交点顺序的正确性,对 B'侧延长线上的交点不保留也不累计;如果被裁剪直线的方向是从 B'到 A',则只累计 B'侧延长线上的交点个数.

裁剪算法是通过交点的顺序来确定直线段在窗口内的部分的.那么,如果出现被裁剪直线或其延长线通过

多边形的一个顶点或与其一边重合的情况(也是上述的后一种情况,即变换后的顶点落在直线 $y=y_c$ 上的情况),则很可能破坏这个规律.这种情况虽然出现的可能性很小,但必须详细地分析.我们将在第 5 节讨论这一问题.

为了验证新方法的高效性,下面将其与现有的两个多边形窗口的线裁剪算法,即 Cyrus-Beck 算法和斜率法进行计算量比较.表 1 列出了 3 种算法在线裁剪操作中对于每一条窗口边进行裁剪操作所需的计算量.可见,新方法的计算量明显小于另外两种方法.

Table 1 Comparison of calculations needed to clip against one edge of the window by the new method and the existing methods

表 1 新方法与现有方法对于一条窗口边进行裁剪操作所需的计算量比较

Edge / Calculations / Methods	Clipping by an edge without intersecting with the line to be clipped				Clipping by an edge intersecting with the line to be clipped			
	Division	Multi-plication	Addition and subtraction	Comparison	Division	Multi-plication	Addition and subtraction	Comparison
Cyrus-Beck method	1	4	6	6	1	4	6	6
The slope method	1	0	2	4	3	2	8	4
The new method	0	1	1	3	1	2	6	3

5 两多边形的边重合或者两多边形在顶点处相交的特殊情况的处理

到此为止,我们假设两多边形的边不重合并且两多边形不在顶点处相交.两多边形的边重合或者两多边形在顶点处相交的特殊情况虽然出现的可能性很小,但必须妥善处理.当前,处理这类特殊情况主要有两种方法.第 1 种方法是文献[13]等描述的精确的处理方法,这种方法将上述的特殊情况分解成各种子情况分别进行处理,通过在链表上增加或减少交点来保持进点和出点是交替出现的;第 2 种方法是由文献[11]提出来的:当遇到上述的特殊情况时,则对重合边的一个顶点或与交点相同的顶点进行很小的移动来避免特殊情况.

上述第 1 种方法虽然很精确,但其处理过程比较复杂,且计算量较大.第 2 种方法的过程简单,且计算量较小,但是不如第 1 种方法精确.然而,由于裁剪算法使用的是浮点数操作,而显示器上的像素却只有整数坐标,所以在显示器上显示的图形实际上就已经具有了一定的误差.该误差一般是小于或等于 0.5.这样,第 2 种方法对顶点进行很小的移动,比如移动的距离是一台计算机所能表示的最小的浮点数(对于单精度浮点类型可取 10^{-37}),则不会对算法的结果产生影响.因此,我们的新算法使用第 2 种方法,只是对其进行了两点改进.

第 1 点改进是只对出现特殊情况的当前边临时改变顶点(对其他边则仍是原顶点),而不像原方法那样永久性地改变该顶点.永久性地改变顶点的方法将可能产生较大的累计误差,因为如果处理后面的边时对于永久性改变了的该顶点又出现了特殊情况,则该顶点就在以前改变的基础上再一次改变,因而可能产生较大的误差.而改进之后该顶点的每一次改变都是在原始顶点的基础上的改变,因而不会产生大的累计误差.第 2 点改进针对顶点的移动方向.文献[11]中没有具体说明如何选择移动顶点的方向.其实如何没有确定的规则而是随意地移动顶点,则移动后的顶点可能仍然没有离开与其重合的边.我们的方法是,如果重合顶点属于裁剪多边形,则只将该顶点的 y 坐标加(或减)一个小量使该顶点上下移动,因为实体多边形的当前边已经错切变换成了水平线,所以不会仍然重合(见第 4 节);如果重合顶点属于实体多边形,则只将该顶点的 x 坐标加(或减)一个小量使该顶点左右移动便可避开裁剪多边形的边.

6 算法比较

这一节我们将新的多边形裁剪算法与 Vatti 算法和 Greiner-Hormann 算法从占用存储空间和所需运行时间两方面进行比较.

首先是空间复杂性分析.由于输出多边形链表对于各算法都是相同的,所以下面的比较不包括输出多边形链表.设实体多边形的顶点数为 n,裁剪多边形的顶点数为 m,两个多边形之间有 k 个交点.

Vatti 算法和 Greiner-Hormann 算法都采用双链表数据结构,所以其空间复杂度是相同的.它们的顶点结点和交点结点的结构是相同的,如下所示:

Vertex={x, y: coordinates;

```
    inters, used: Boolean;
    next, prev: pointer;
    entryexit: Boolean;
    neighber: pointer;
    nextpoly: pointer;
  }
```

如果以结点的一个域为一个存储单位,则上面结构的每一个结点需要 9 个存储单位.由于每求出一个交点都需要建立两个结点,所以算法所需的总的存储单位为

$$9(n+m+2k).$$

对于新算法,由于采用单链表结构而无须向前的指针及进出点标记等,所以,每一个顶点结点需要 5 个存储单位(除第 1 个顶点结点之外);每一个交点结点需要 6 个存储单位(见第 2 节).新算法的顶点结点的数量为 $n+m$;由于新算法每求出一个交点只建立一个结点,所以交点结点的数量为 k.因此,新算法所需的总的存储单位为

$$5(n+m)+6k.$$

显然,新算法所需的总的存储单位远远小于 Vatti 算法和 Greiner-Hormann 算法.我们可以计算出后者比新算法多占用的存储单位是

$$9(n+m+2k)-[5(n+m)+6k]=4(n+m)+12k.$$

可见,新算法几乎少用了一半的存储单位.

下面比较各算法所需的计算量,它决定了算法的运行速度.

首先,新算法的简单性使得与 Vatti 算法和 Greiner-Hormann 算法相比减少了许多操作.新算法将交点的计算与进出性判断合成一步进行,而省略了单独地判断一点是否在多边形内的过程.从而少了一次对多边形链表的遍历.另外,利用了两个多边形之间进、出点之间的关系及在一个多边形上进、出点之间的分布关系,所以新算法只需标记第 1 个交点是进点还是出点,其他交点的进出性则可依次确定.而 Vatti 算法和 Greiner-Hormann 算法使用双链表数据结构,有时需要反向遍历链表,所以其进、出点之间的分布关系比较复杂.因此需要判断并存储每个交点的进出性,这需要较多的操作.新算法虽然有时需要一次对裁剪多边形链表的遍历来反向该链表,但这却使采用单链表成为可能,进而节省了大量的操作.人们的输入习惯和程序设计习惯往往是以相同的方向输入和编程处理多边形,所以反向裁剪多边形链表的概率应该是比较小的.

其次,新算法简单的数据结构和结点结构不仅节省了大量的存储空间,而且当遍历进行到每个结点时,由于不需要修改和设置较多的域而节省了大量的时间.

再次,本文提出的算法所使用的判断和计算交点的新方法大幅度减少了所需的计算量,见上一节中的表 1.由于在 Vatti 的论文[10]和 Greiner 及 Hormann 的论文[11]中没有介绍所使用的判断和计算交点的方法,所以我们认为他们所使用的是已有的且较好的判断和计算交点的方法.因此,表 1 中将新方法与已有的且较好的判断和计算交点的方法进行了比较.

最后,我们对新算法、Vatti 算法和 Greiner-Hormann 算法进行实际运行时间比较.我们用 C 语言在主频为 266MHz 的微机上实现了这 3 个算法.为了使比较结果更简洁、直观,我们随机给出有 n 个顶点的两个多边形,并测量各算法对它们做裁剪操作所需的时间.表 2 列出了对于不同的 n 值,各算法进行 1 000 次裁剪操作所需时间的比较.可见,新算法所需的时间明显少于 Vatti 算法和 Greiner-Hormann 算法.

7 结 论

本文提出了一个有效的多边形裁剪的算法,它不仅适用于凹多边形,而且适用于带孔洞的多边形.该算法不仅可以求多边形的“交”(多边形裁剪),而且也可以求多边形的“并”和“差”.另外,还适用于有多个分立的结果多边形的情况,具有一般性.整个算法采用单链表作为输入输出的数据结构,且无须限制输入多边形的方向.本文还提出了一些新的技术和方法作为该算法的基础.最后,将新算法与最新的和最有效的两个同类算法进行了比较.比较的结果表明,新算法无论在占用存储空间方面,还是在计算量和运行速度方面,都优于后两者.

Table 2 Comparison of the running times for 1 000 clipping operations by the new algorithm, the Vatti algorithm and the Greiner-Hormann algorithm (s)

表 2 新算法与 Vatti 算法和 Greiner-Hormann 算法执行 1 000 次裁剪操作所需的时间比较 (秒)

N	Vatti algorithm	Greiner-Hormann algorithm	New algorithm		
			Two polygons have the same edge direction	Two polygons have different edge directions	Average
3	0.255 61	0.164 83	0.137 24	0.137 48	0.137 36
5	0.628 93	0.357 14	0.274 72	0.302 20	0.288 46
10	2.472 05	1.373 63	1.181 32	1.208 79	1.195 05
20	6.845 91	3.475 27	2.912 09	3.021 98	2.967 03
50	34.714 3	16.071 43	12.912 08	14.835 16	13.873 62

References:

[1] Sutherland IE, Hodgeman GW. Reentrant polygon clipping. Communications of the ACM, 1974,17(1):32~42.

[2] Liang Y, Barsky BA. An analysis and algorithm for polygon clipping. Communications of the ACM, 1983,26(11):868~877.

[3] Foley JD, Dam A, Feiner SK, Hughes JF. Computer Graphics, Principles And Practice. Reading, MA: Addison-Wesley, 1990.

[4] Maillot PG. A new, fast method for 2D polygon clipping: Analysis and software implementation. ACM Transactions on Graphics, 1992,11(3):276~290.

[5] Andereev RD. Algorithm for clipping arbitrary polygons. Computer Graphics Forum, 1989,8(2):183~191.

[6] Montani C, Re M. Vector and raster hidden surface removal using parallel connected stripes. IEEE Computer Graphics and Applications, 1987,7(1):14~23.

[7] Reppaport A. An efficient algorithm for line and polygon clipping. Visual Computer, 1991,7(1):19~28.

[8] Sechrest S, Greenberg D. A visible polygon reconstruction algorithm. Computer Graphics, 1981,15(1):17~26.

[9] Weiler K, Atherton P. Hidden surface removal using polygon area sorting. In: Proceedings of the SIGGRAPH'77. New York: ACM Press, 1977. 214~222.

[10] Vatti BR. A generic solution to polygon clipping. Communications of the ACM, 1992,35(1):56~63.

[11] Greiner G, Hormann K. Efficient clipping of arbitrary polygons. ACM Transactions on Graphics, 1998,17(2):71~83.

[12] Cyrus M, Beck J. Generalized two- and three- dimensional clipping. Computer and Graphics, 1978,3(1):23~28.

[13] Liu YK, Yan Y, Shi JY. An efficient algorithm for line clipping against a polygon. Chinese Journal of Computers, 1999,22(11): 1209~1214 (in Chinese with English abstract).

附中文参考文献:

[13] 刘勇奎,颜叶,石教英.一个有效的多边形窗口的线裁剪算法.计算机学报,1999,22(11):1209~1214.